Calcolatori Elettronici A
a.a. 2008/2009

RETI LOGICHE:
RETI COMBINATORIE

Massimiliano Giacomin

Reti combinatorie

DEFINIZIONE

- Una rete combinatoria è un circuito elettronico in grado di ‘***calcolare’ in modo automatico funzioni binarie di una o più variabili binarie***
- Le **uscite** di una rete combinatoria dipendono unicamente dai **valori di ingresso**

ELEMENTI COSTITUTIVI

–un gruppo di elementi attivi: le **porte logiche**

–collegati fra loro da elementi passivi: **linee**

•di ingresso: solo un’estremità collegata all’ingresso di una porta

•di uscita: solo un’estremità è collegata all’uscita di una porta

•di circuito: collegano l’uscita di una porta con l’ingresso di una diversa porta

Specifica del comportamento di una rete combinatoria

- Significa specificare per ogni uscita una ***funzione booleana***

$$f(x_0,\ldots, x_{n-1})$$

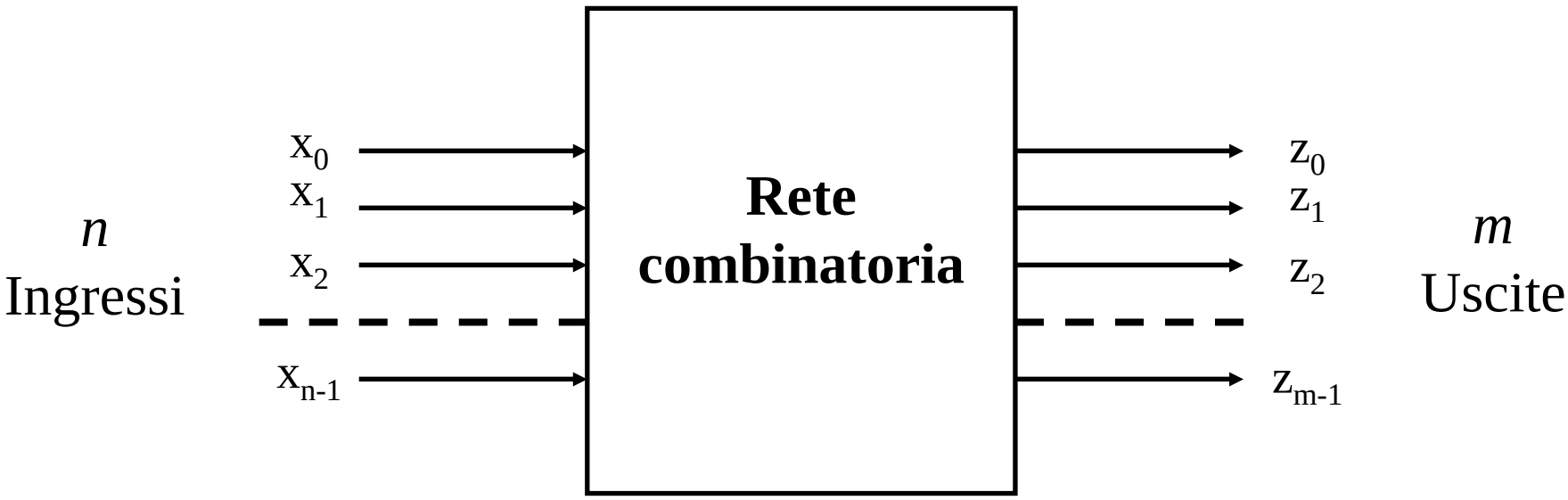
$$f : \{0,1\}^n \rightarrow \{0,1\}$$

- Esistono due modi per specificare una funzione booleana:
  - ***tabella di verità***

    elenca, per ognuna delle 2^n possibili combinazioni (dei valori) degli ingressi, il valore corrispondente della funzione
  - ***espressione booleana***:

    utilizzando operatori logici, esprime il legame tra le variabili di uscita (corrispondenti alle uscite) e le variabili di ingresso (corrispondenti agli ingressi)

Rete combinatoria come “scatola nera”



Il comportamento della rete è descritto da:

- tabella di verità (*n* ingressi, *m* colonne per le uscite)
- *m* espressioni booleane in *n* variabili

ESEMPI DI TABELLE DI VERITA'

x_1	x_0	$x_1 \bullet x_0$
0	0	0
0	1	0
1	0	0
1	1	1

AND

x_1	x_0	$x_1 + x_0$
0	0	0
0	1	1
1	0	1
1	1	1

OR

x	$\overline{x}$
0	1
1	0

NOT

ESEMPI DI TABELLE DI VERITA’

x_1	x_0	$\overline{x_1 \bullet x_0}$
0	0	1
0	1	1
1	0	1
1	1	0

NAND

x_1	x_0	$\overline{x_1 + x_0}$
0	0	1
0	1	0
1	0	0
1	1	0

NOR

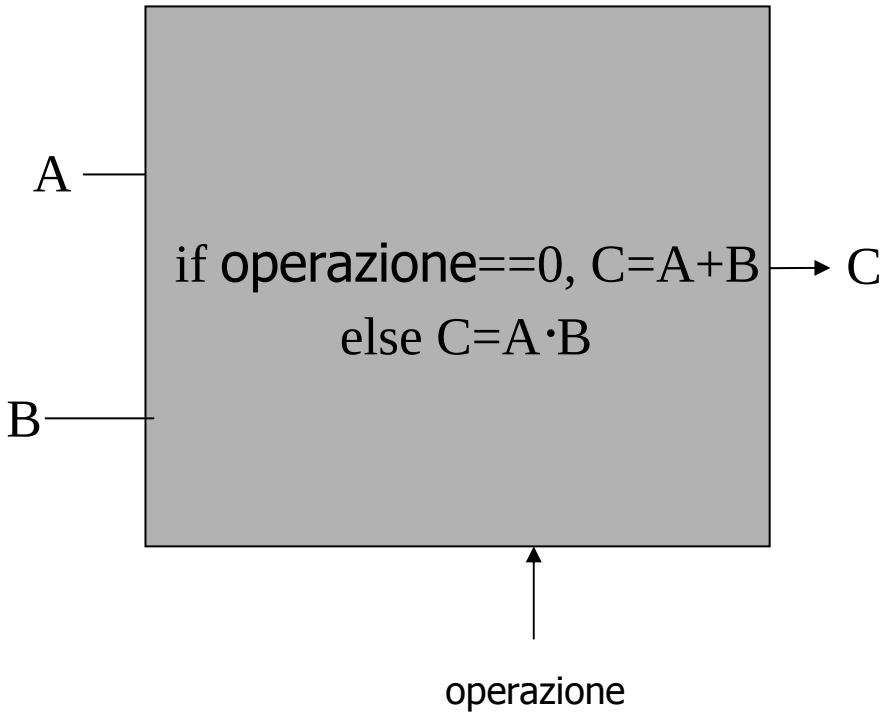
Dato n, 2^{2^n} funzioni booleane di *n* variabili

Esempio con $n = 2$

x_1	x_0	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

AND OR NOR ≡ NAND

ESEMPIO DI SPECIFICA DI UNA RETE COMBINATORIA CON TABELLA DI VERITA'



operazione	***A***	***B***	***C***
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Espressioni (formule) booleane

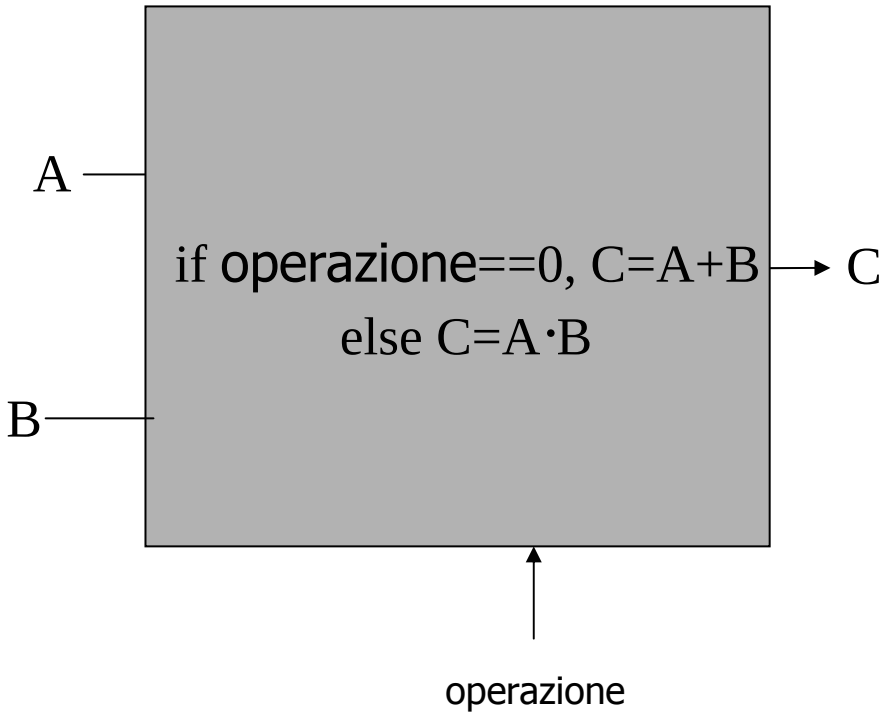
1. Le **costanti 0** e **1** e le **variabili** (simboli a cui possono essere associati i valori 0 e 1) sono espressioni booleane
2. Se E, E_1 ed E_2 sono espressioni booleane lo sono anche **(E_1+E_2), $(E_1 \cdot E_2)$** e **$(\overline{E})$**
3. Non esistono altre espressioni oltre a quelle che possono essere generate da un numero finito di applicazioni delle regole 1 e 2

ESEMPI

- $((x1+x2)\cdot x3)$
- $((x1\cdot x2)+(x3\cdot(x4+x5)))$

NB: come nelle espressioni aritmetiche, $\cdot$ ha priorità su +

ESEMPIO DI SPECIFICA DI UNA RETE COMBINATORIA CON ESPRESSIONE BOOLEANA



$$C = \overline{\text{operazione}} \cdot (A+B) + \text{operazione} \cdot A \cdot B$$

Algebra di Boole

- E’ lo **strumento matematico** usato per lo studio delle reti combinatorie espresse mediante formule booleane
- E’ un particolare tipo di algebra che include:
  - un insieme di supporto **A** (l’insieme **{0,1}** nel ns caso)
  - degli operatori binari: **AND** (·) e **OR** (+)
  - un operatore complemento: **NOT** ($\bar{}$)
- Gli operatori soddisfano certe **proprietà** che si deducono da un insieme di **assiomi**

Assiomi e alcune proprietà dell'Algebra di Boole

		Forma AND	**Forma OR**
Assiomi	Commutatività	$AB = BA$	$A+B = B+A$
	Distributività	$A+BC=(A+B)(A+C)$	$A(B+C)=AB+AC$
	Identità	$1A = A$	$0+A = A$
	Inverso	$A\bar{A} = 0$	$A+\bar{A} = 1$
		dualità	
Proprietà	Elem. nullo	$0A = 0$	$1+A = 1$
	Idempotenza	$AA = A$	$A+A = A$
	Assorbimento	$A(A+B) = A$	$A+AB=A$
	Associatività	$(AB)C = A(BC)$	$(A+B)+C=A+(B+C)$
	De Morgan	$\overline{AB} = \overline{A}+\overline{B}$	$\overline{A+B} = \overline{A}\,\overline{B}$

Tabelle di verità e proprietà dell'Algebra di Boole: Esempio

Proprietà di De Morgan: $\overline{x_1 x_0} = \overline{x_1} + \overline{x_0}$

x_1	x_0	$x_1 x_0$	$\overline{x_1 x_0}$	$\overline{x_1}$	$\overline{x_0}$	$\overline{x_1} + \overline{x_0}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Usare l’algebra di Boole per dimostrare l’equivalenza fra formule booleane: alcuni esempi

- $x_1x_2 + x_1\overline{x_2}x_3 = x_1\,(x_2+\overline{x_2}x_3)$
- $x_1 + x_2 + x_2x_3 + \overline{x_2}x_3 = x_1 + x_2 + x_3(x_2+\overline{x_2}) = x_1 + x_2 + x_3$
- $x_1x_2 + x_1x_2x_3 + x_1x_2 = x_1x_2 + x_1x_2x_3 = x_1x_2\,(1 + x_3) = x_1x_2$

Espressioni e funzioni booleane

- Ad una espressione booleana di *n* variabili corrisponde un'**unica** funzione booleana di *n* variabili (cioè una tabella)
- Viceversa, ad una funzione booleana di *n* variabili corrispondono **infinite** espressioni booleane di *n* variabili (cioè se partiamo da una tabella scopriamo che vi sono infinite espressioni tra loro equivalenti)
  - Alcuni esempi: vedi lucido precedente
  - Altri esempi: vedi lucido seguente

Rappresentazioni canoniche

- Tra le diverse espressioni booleane equivalenti corrispondenti a una stessa funzione se ne individuano due, chiamate:
  - Forma canonica disgiuntiva (o “**somma di prodotti**”)
  - Forma canonica congiuntiva (o “**prodotto di somme**”)

- Nel seguito, faremo ampio uso della prima

Esempio: forma canonica disgiuntiva di una funzione booleana a 3 variabili

x_2	x_1	x_0	$f(x_0, x_1, x_2)$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

$$f = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1\overline{x}_0 + x_2\overline{x}_1x_0$$

Mintermini

- ***mintermine***: funzione booleana che assume il valore 1 in corrispondenza di una e una sola configurazione degli ingressi

Esempio con 3 variabili

$\bar{x}_2\bar{x}_1\bar{x}_0$	m_0	(000)
$\bar{x}_2\bar{x}_1 x_0$	m_1	(001)
$\bar{x}_2 x_1\bar{x}_0$	m_2	(010)
$\bar{x}_2 x_1 x_0$	m_3	(011)
$x_2\bar{x}_1\bar{x}_0$	m_4	(100)
$x_2\bar{x}_1 x_0$	m_5	(101)
$x_2 x_1\bar{x}_0$	m_6	(110)
$x_2 x_1 x_0$	m_7	(111)

Forma canonica disgiuntiva “somma di prodotti”

Quindi, la formula precedente

$$f = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1\overline{x}_0 + x_2\overline{x}_1x_0$$

si può scrivere come

$$\boldsymbol{f = m_1 + m_2 + m_5}$$

SINTESI DI RETI LOGICHE COMBINATORIE

- Finora abbiamo parlato di come specificare il comportamento di una rete combinatoria, senza preoccuparci di “come è fatta”
- A questo livello, “come è fatta” una rete logica significa “da quali porte logiche è fatta” e come queste sono collegate
- ***Sintesi***: data una funzione logica, come progettare una rete combinatoria che la calcola?
- Vedremo come sintetizzare una rete:
  - utilizzando una combinazione di porte logiche
  - utilizzando PLA (Programmable Logic Array)
  - utilizzando una ROM (Read Only Memory)

SINTESI DI RETI LOGICHE COMBINATORIE

- Finora abbiamo parlato di come specificare il comportamento di una rete combinatoria, senza preoccuparci di "come è fatta"
- A questo livello, "come è fatta" una rete logica significa "da quali porte logiche è fatta" e come queste sono collegate
- ***Sintesi***: data una funzione logica, come progettare una rete combinatoria che la calcola?
- Vedremo come sintetizzare una rete:
  - utilizzando una combinazione di porte logiche
  - utilizzando PLA (Programmable Logic Array)
  - utilizzando una ROM (Read Only Memory)

Per progettare una rete combinatoria:

1. Individuare le **variabili** di ingresso e di uscita e la **tabella** di verità
2. Derivare dalla tabella una **espressione booleana** (ad esempio quella in forma canonica disgiuntiva) per ognuna delle *m* linee di uscita
3. Costruire la **rete** combinatoria associata alle *m* espressioni booleane

Esempio: funzione di maggioranza

x_2	x_1	x_0	$f(x_0, x_1, x_2)$
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

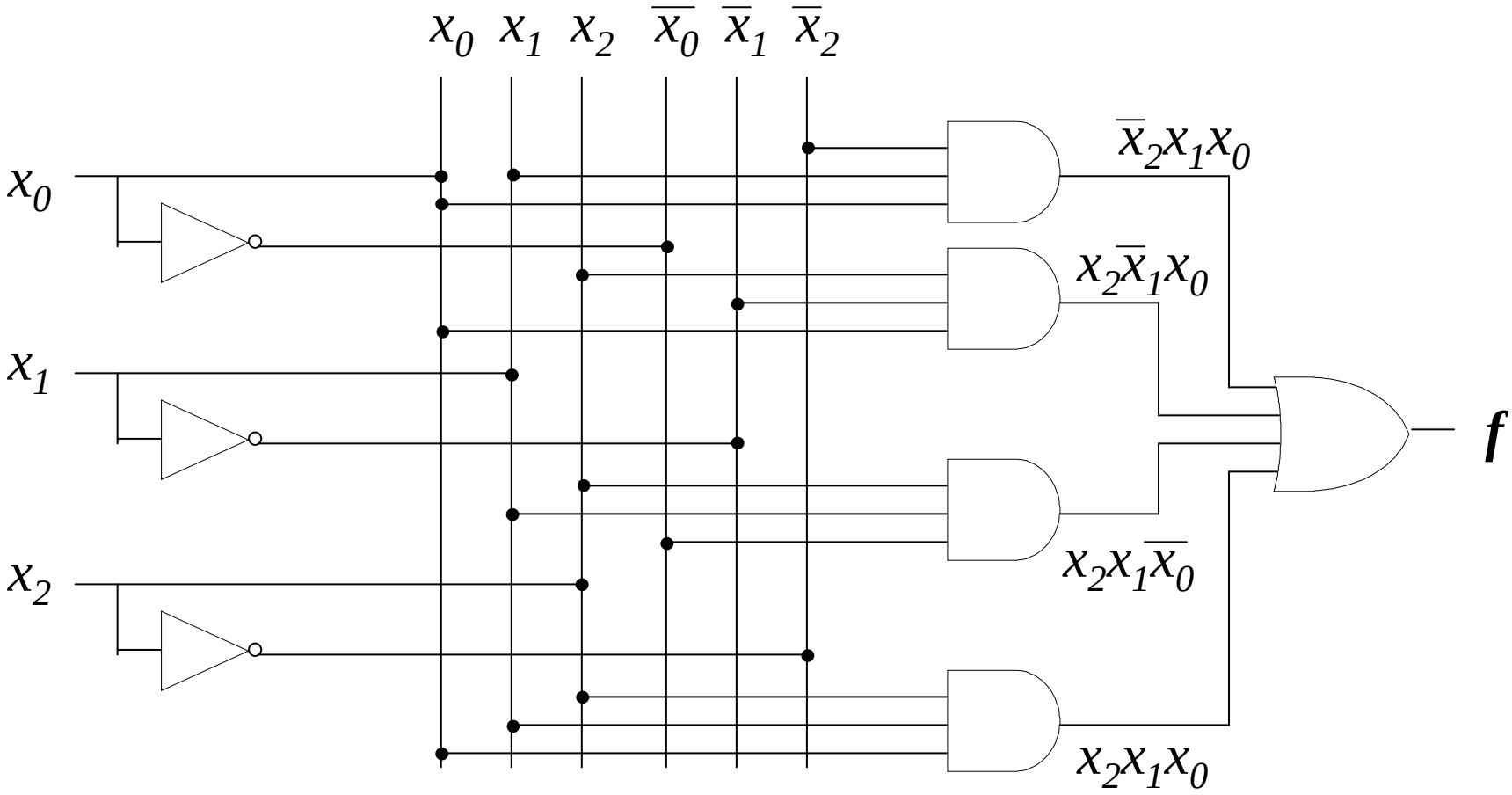
$n = 3$ ingressi

$m = 1$ uscite

$$f(x_0, x_1, x_2) = \overline{x}_2 x_1 x_0 + x_2 \overline{x}_1 x_0 + x_2 x_1 \overline{x}_0 + x_2 x_1 x_0$$

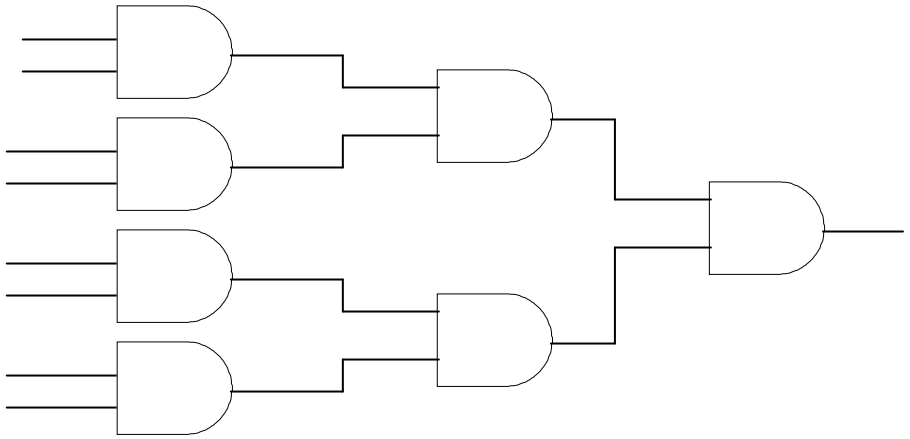
Attenzione: la formula *f* si può semplificare!

Sintesi della funzione di maggioranza



Nota: porte a k > 2 ingressi

- Come visto nell'es. precedente spesso è preferibile disporre di **porte con un numero arbitrario k di ingressi**
- Se si hanno a disposizione solo **porte a 2 ingressi**, le porte a k ingressi vengono realizzate tramite alberi binari di porte a 2 ingressi



Funzione AND con 8 linee di ingresso realizzata con porte AND a 2 ingressi

Sintesi utilizzando solo porte NAND o NOR

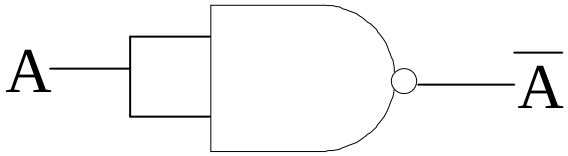
- Come visto, qualunque rete combinatoria (funzione logica) può essere sintetizzata con sole porte AND, OR, NOT
- Si può dimostrare che tutte le funzione logiche possono essere realizzate con un solo tipo di porta logica, a patto che comprenda un'inversione: queste porte vengono dette **universali**
- Spesso la tecnologia (livello circuitale) mette a disposizione solo una porta invertente:

  NAND (porta AND con uscita invertita)

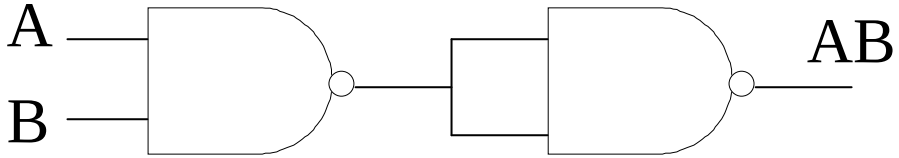
  NOR (porta OR con uscita invertita)

  Dimostriamo che NAND e NOR sono universali…

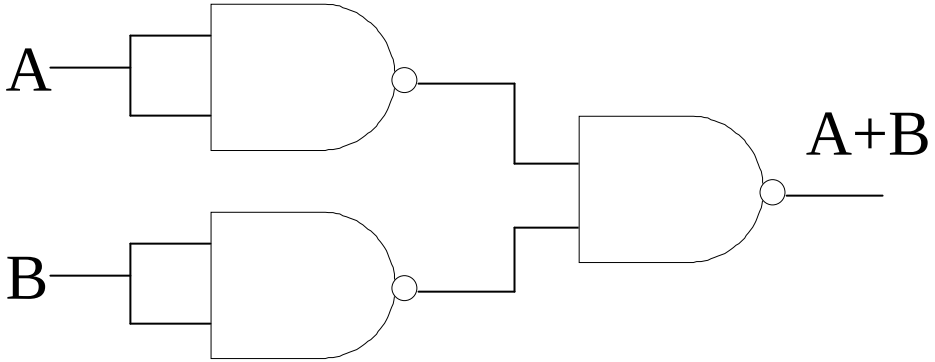
Porte NOT, AND, OR usando solo porte NAND



NOT



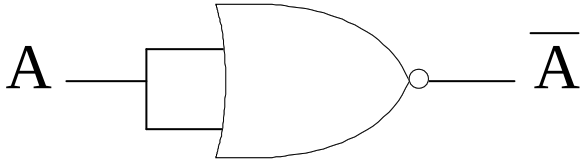
AND



OR

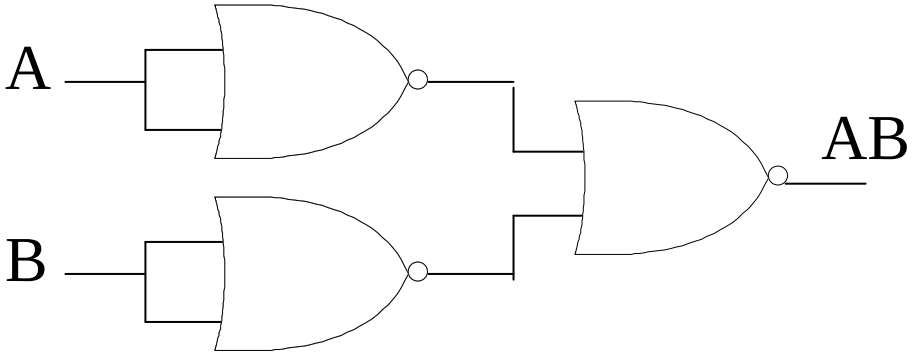
$$\left[A+B=\overline{\overline{A+B}}=\overline{\overline{A}\,\overline{B}} \right]$$

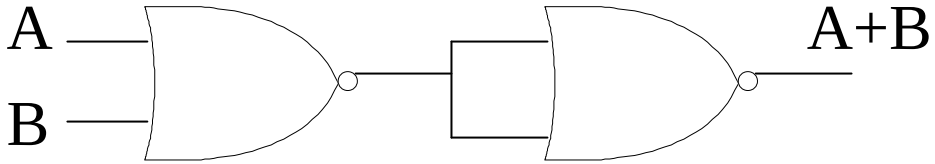
Porte NOT, AND, OR usando solo porte NOR



NOT

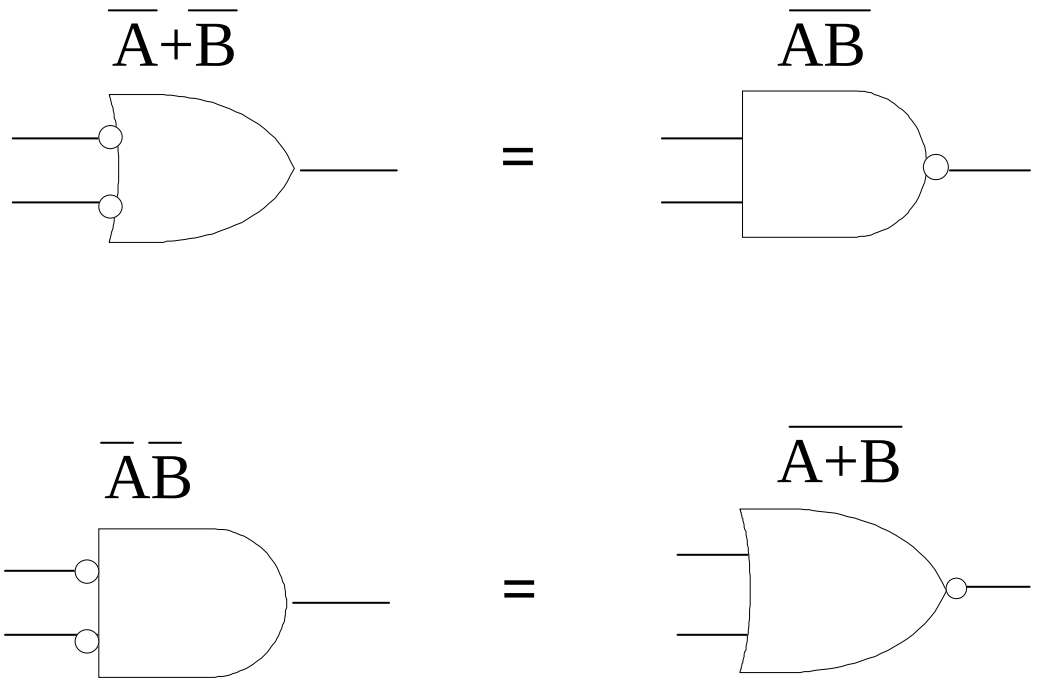
AND





OR

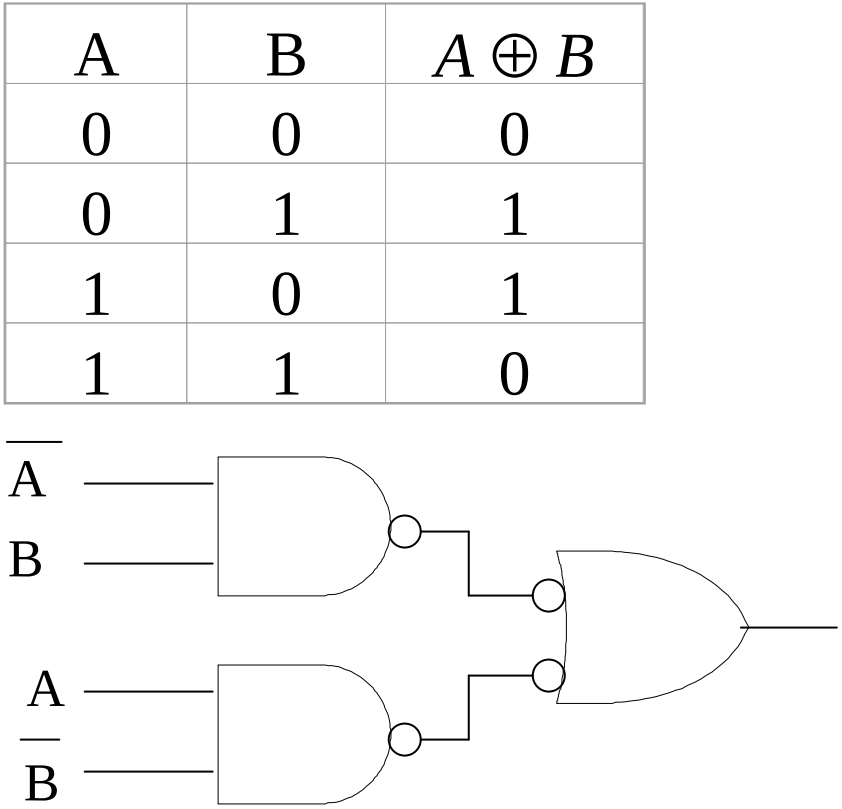
Equivalenze grafiche (per ricondursi a NAND o NOR)

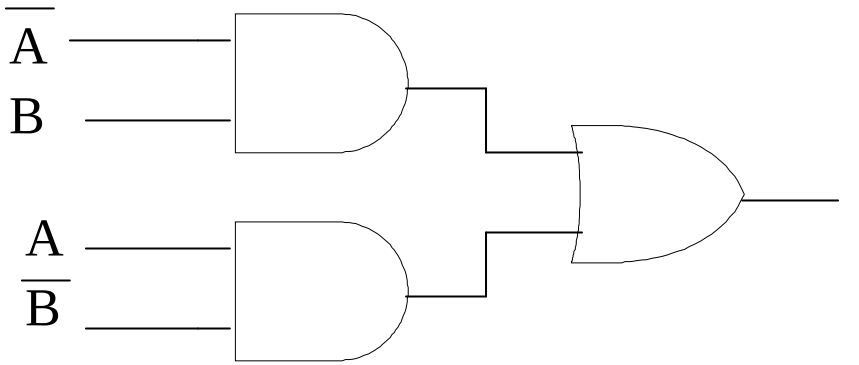


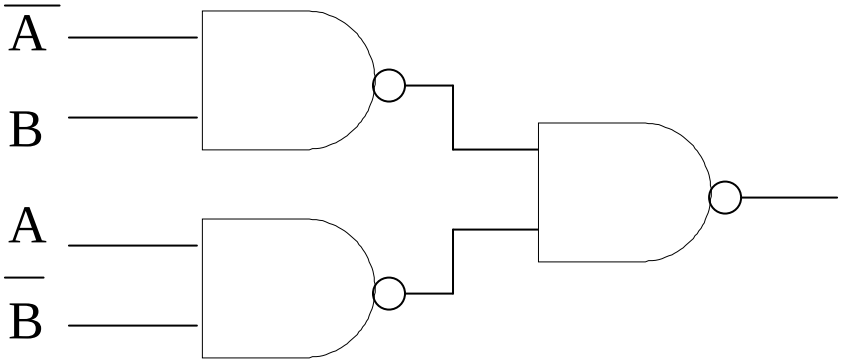
NB: oppure si possono usare le proprietà dell'algebra booleana
(in particolare la proprietà di De Morgan)

Es: la funzione XOR con porte NAND

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

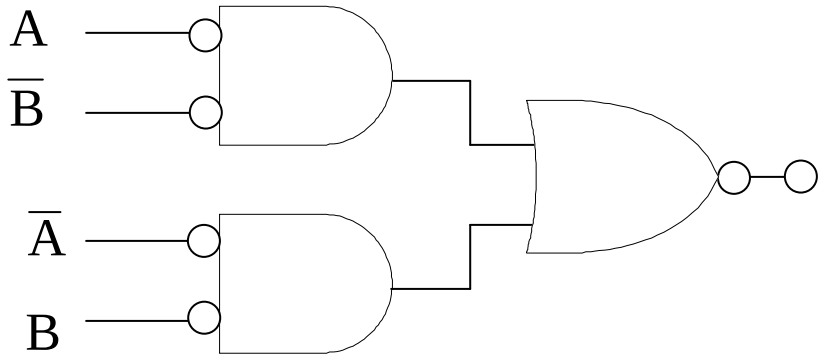


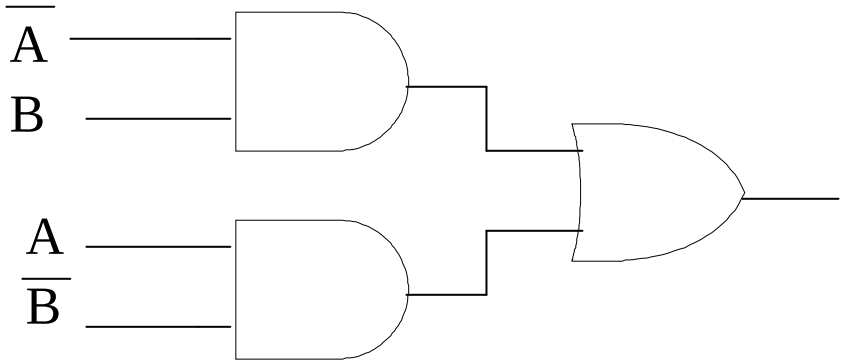


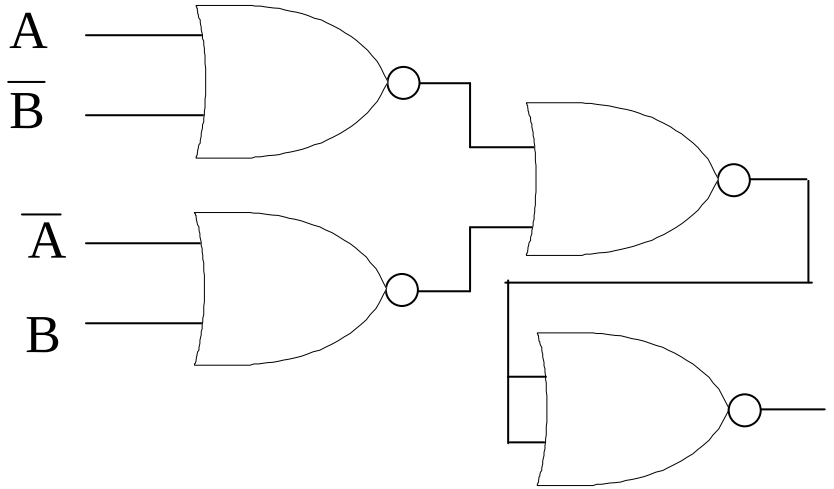


Es: la funzione XOR con porte NOR

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0







Attenzione alle variabili negate!!!

Minimizzazione (cenni)

- Data una funzione da realizzare: derivare l'espressione "minima":
  - l'espressione che porta ad una realizzazione con il numero minimo di elementi di calcolo
  - dipende dalla tecnologia
- Esistono tecniche di minimizzazione (mappe di Karnaugh, metodo di Quine-McCluskey, ecc.)
- In pratica, si usano tool automatici di minimizzazione:
  - dalla specifica (linguaggio di descrizione dell'hardware)
  - all'implementazione (fino a livello del layout)

NOI: abbiamo a disposizione le proprietà dell'algebra booleana

Condizioni di indifferenza

- Può accadere che:
  - per alcune configurazioni di ingresso qualsiasi valore delle uscite vada bene
  - alcune configurazioni di ingresso non si presentino mai e quindi per queste qualsiasi valore delle uscite va bene
- I valori delle uscite per queste configurazioni vengono chiamati **valori di indifferenza** e indicati con **x** oppure **d**
- I valori di indifferenza possono giocare un ruolo nella minimizzazione della funzione
- Esempio: **codifica BCD** (Binary Coded Decimal):
  - rappresenta le **cifre decimali** mediante gruppi di **4 bit**…
  - … con 4 bit si ottengono **16** configurazioni di ingresso ma ne bastano **10** per rappresentare i numeri da 0 a 9

Esempio della codifica BCD

Cifra decimale rappresentata	**Codifica binaria**			
	b_3	b_2	b_1	b_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
	1	0	1	0
	1	0	1	1
	1	1	0	0
	1	1	0	1
	1	1	1	0
	1	1	1	1

Vogliamo f=1 per i multipli di 3 escluso lo 0

Esempio della codifica BCD

Cifra decimale rappresentata	**Codifica binaria**				***f***
	b_3	b_2	b_1	b_0	
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	1
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	1
	1	0	1	0	x
	1	0	1	1	x
	1	1	0	0	x
	1	1	0	1	x
	1	1	1	0	x
	1	1	1	1	x

Uso delle condizioni di indifferenza per la minimizzazione

- I valori della funzione corrispondenti alle condizioni di indifferenza possono essere considerati 0 o 1, ciò permette di semplificare le espressioni

Cifra decimale rappresentata	Codifica binaria b_3	b_2	b_1	b_0	f	
0	0	0	0	0	0	
1	0	0	0	1	0	
2	0	0	1	0	0	
3	0	0	1	1	1	
4	0	1	0	0	0	
5	0	1	0	1	0	
6	0	1	1	0	1	
7	0	1	1	1	0	
8	1	0	0	0	0	
9	1	0	0	1	1	
	1	0	1	0	x	
	1	0	1	1	x	← a
	1	1	0	0	x	
	1	1	0	1	x	← b
	1	1	1	0	x	← c
	1	1	1	1	x	← d

$$f = \bar{b}_3\bar{b}_2 b_1 b_0 + \bar{b}_3 b_2 b_1 \bar{b}_0 + b_3\bar{b}_2\bar{b}_1 b_0 =$$

(e) (f) (g)

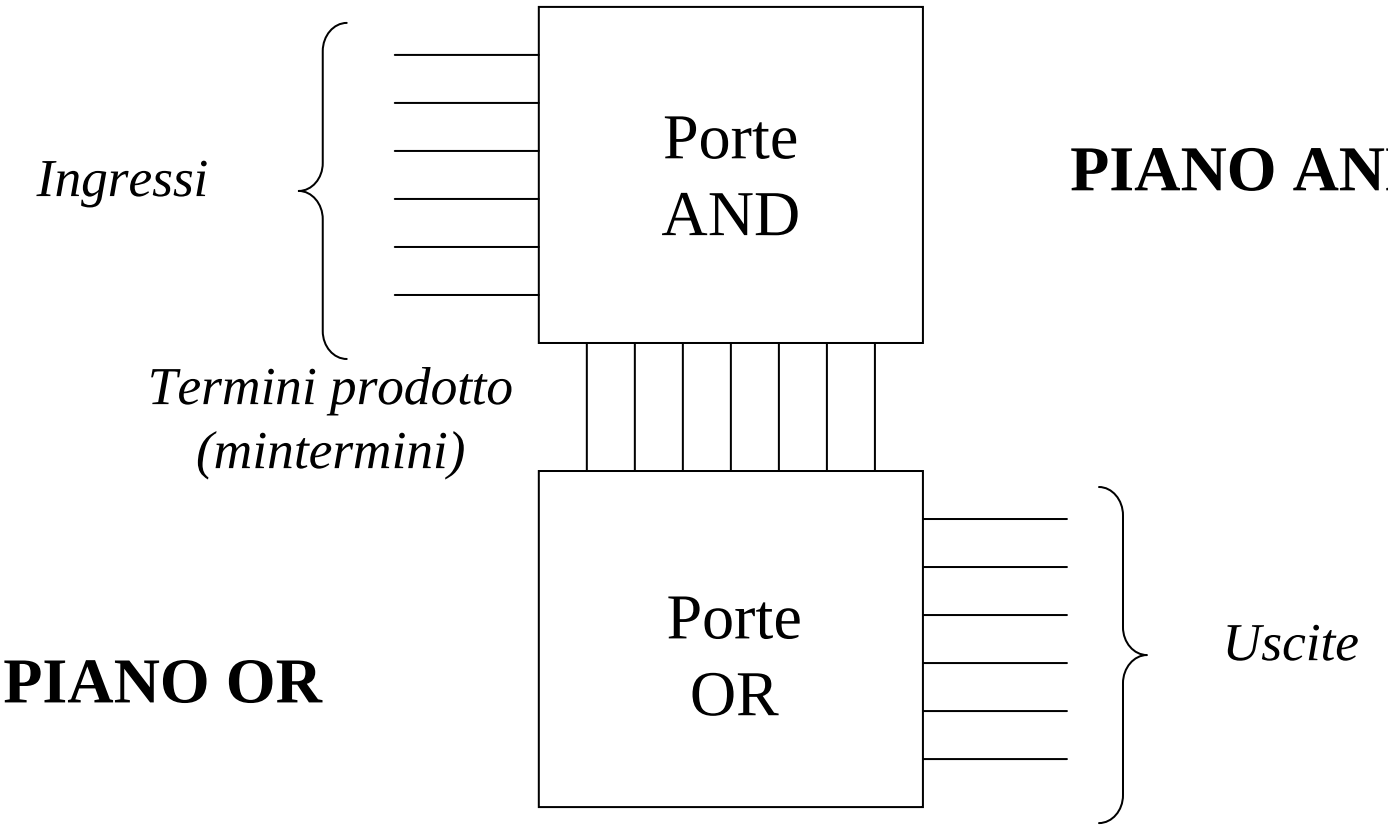
$$\bar{b}_2 b_1 b_0 + b_2 b_1 \bar{b}_0 + b_3 b_0$$

(e+a) (f+c) (g+a+b+d)

SINTESI DI RETI LOGICHE COMBINATORIE

- Finora abbiamo parlato di come specificare il comportamento di una rete combinatoria, senza preoccuparci di “come è fatta”
- A questo livello, “come è fatta” una rete logica significa “da quali porte logiche è fatta” e come queste sono collegate
- ***Sintesi***: data una funzione logica, come progettare una rete combinatoria che la calcola?
- Vedremo come sintetizzare una rete:
  - utilizzando una combinazione di porte logiche
  - utilizzando PLA (Programmable Logic Array)
  - utilizzando una ROM (Read Only Memory)

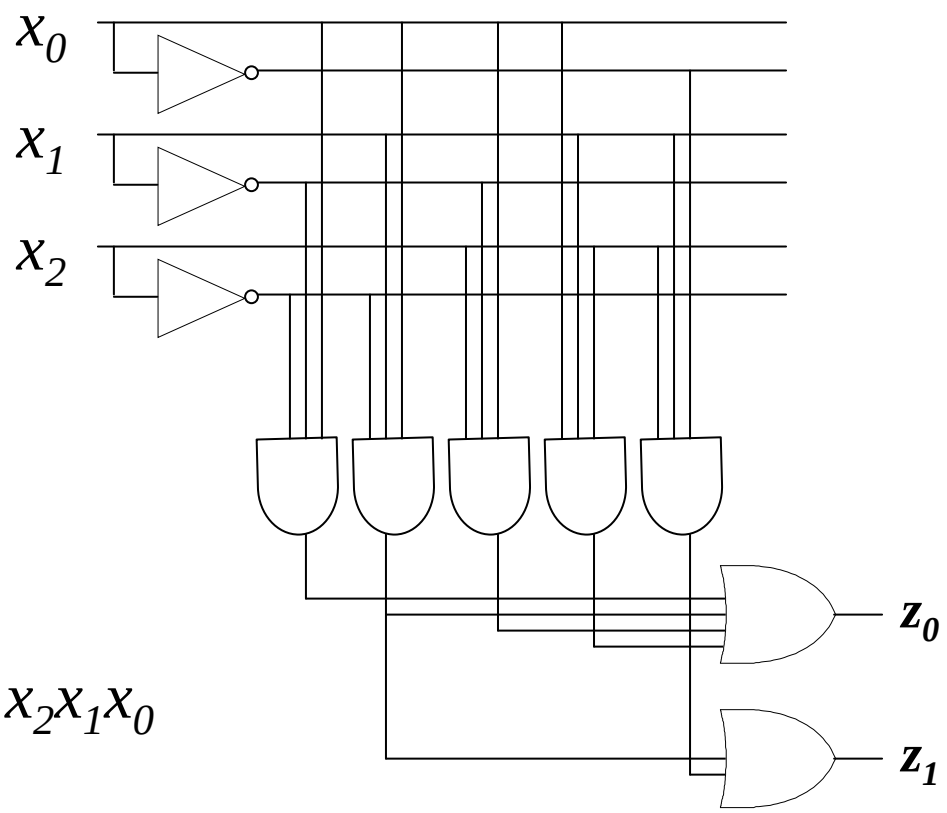
- **PLA**: tecnologia di implementazione che permette di realizzare direttamente funzioni logiche a partire da una tabella di verità (logica strutturata)



Dimensione: n*P+ P*m

Esempio di PLA

x_2	x_1	x_0	z_1	z_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1

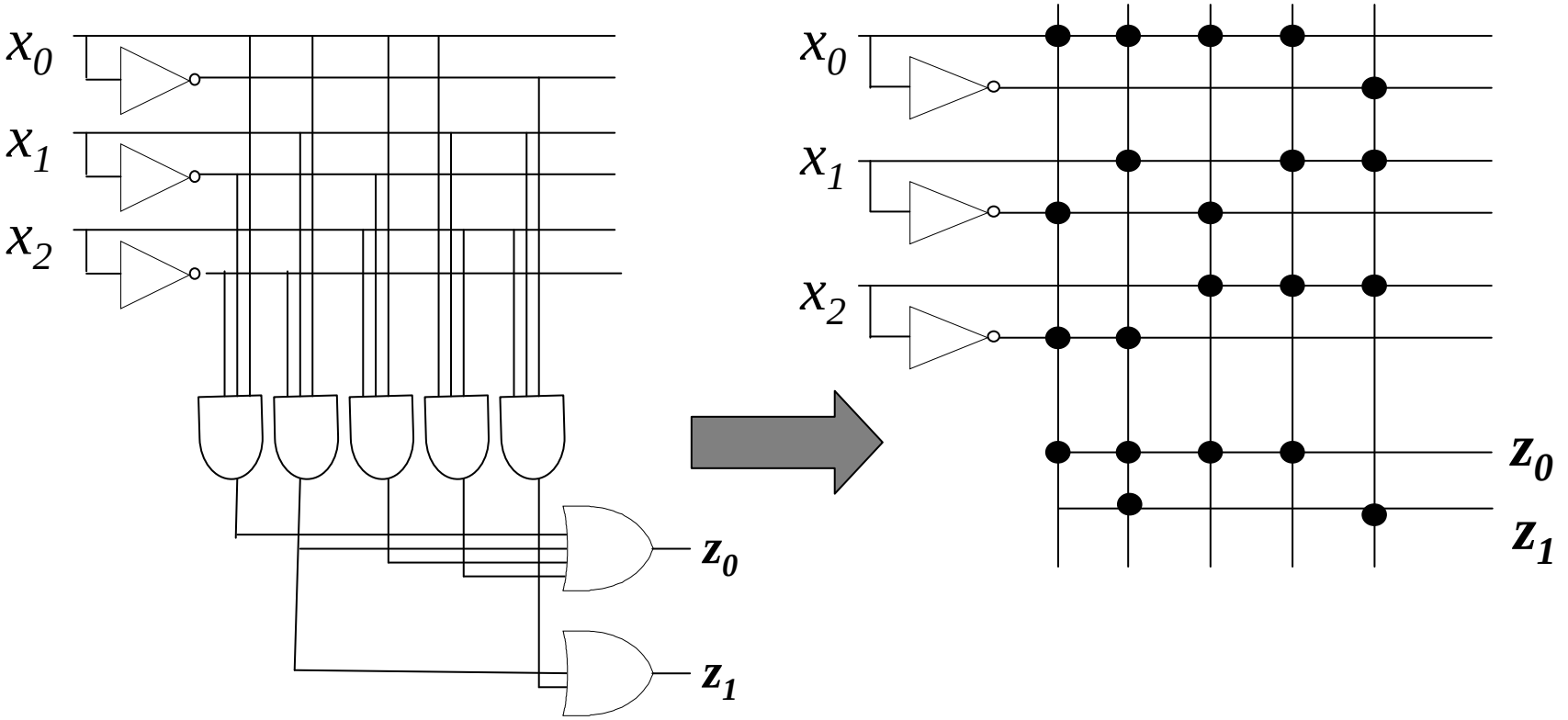


$$z_0 = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1x_0 + x_2\overline{x}_1x_0 + x_2x_1x_0$$

$$z_1 = \overline{x}_2x_1x_0 + x_2x_1\overline{x}_0$$

- 5 AND per i mintermini necessari
- 2 OR perché 2 sono le uscite

Un altro tipo di rappresentazione per le PLA

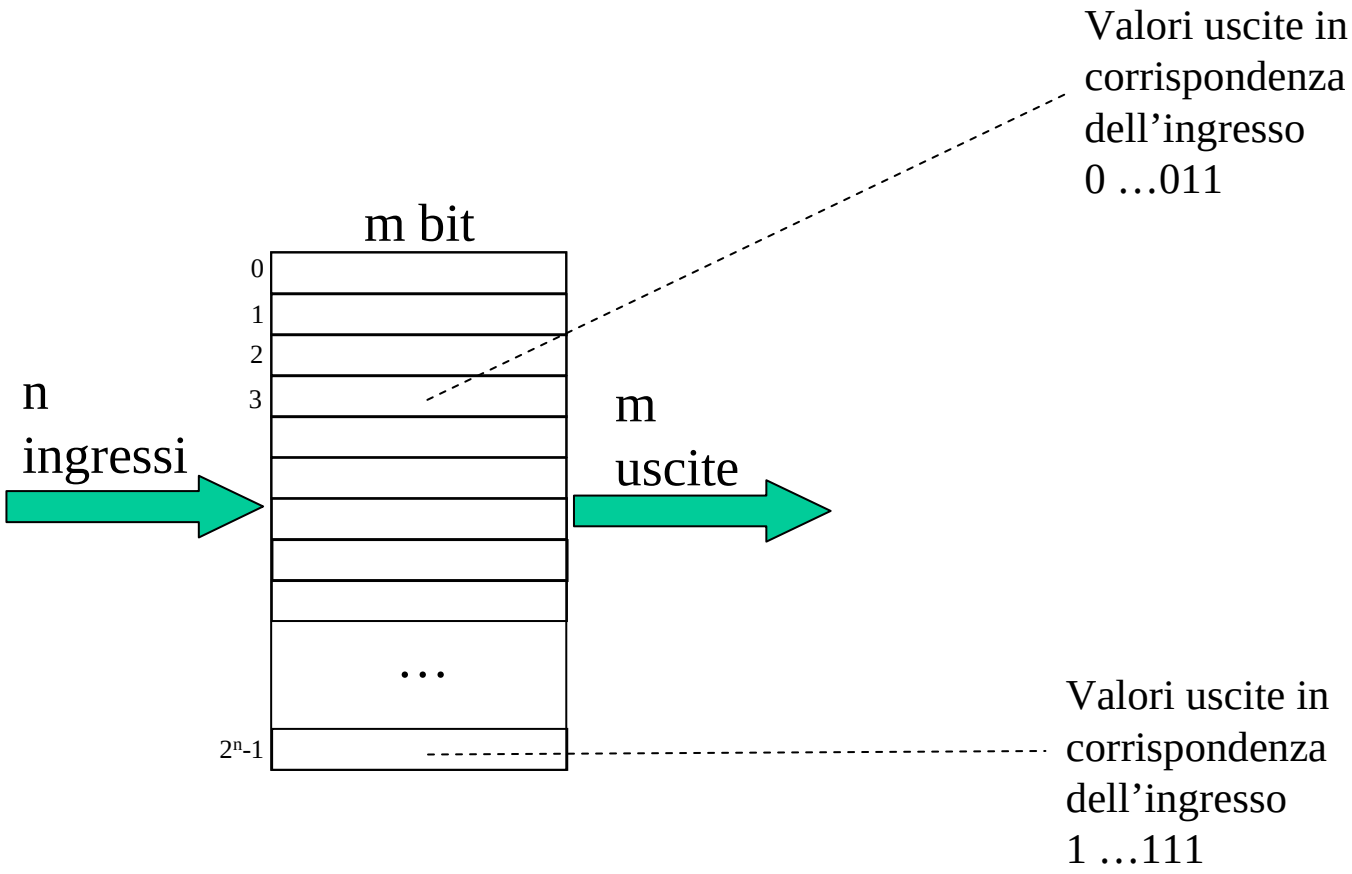


SINTESI DI RETI LOGICHE COMBINATORIE

- Finora abbiamo parlato di come specificare il comportamento di una rete combinatoria, senza preoccuparci di "come è fatta"
- A questo livello, "come è fatta" una rete logica significa "da quali porte logiche è fatta" e come queste sono collegate
- ***Sintesi***: data una funzione logica, come progettare una rete combinatoria che la calcola?
- Vedremo come sintetizzare una rete:
  - utilizzando una combinazione di porte logiche
  - utilizzando PLA (Programmable Logic Array)
  - utilizzando una ROM (Read Only Memory)

ROM (Read Only Memory)

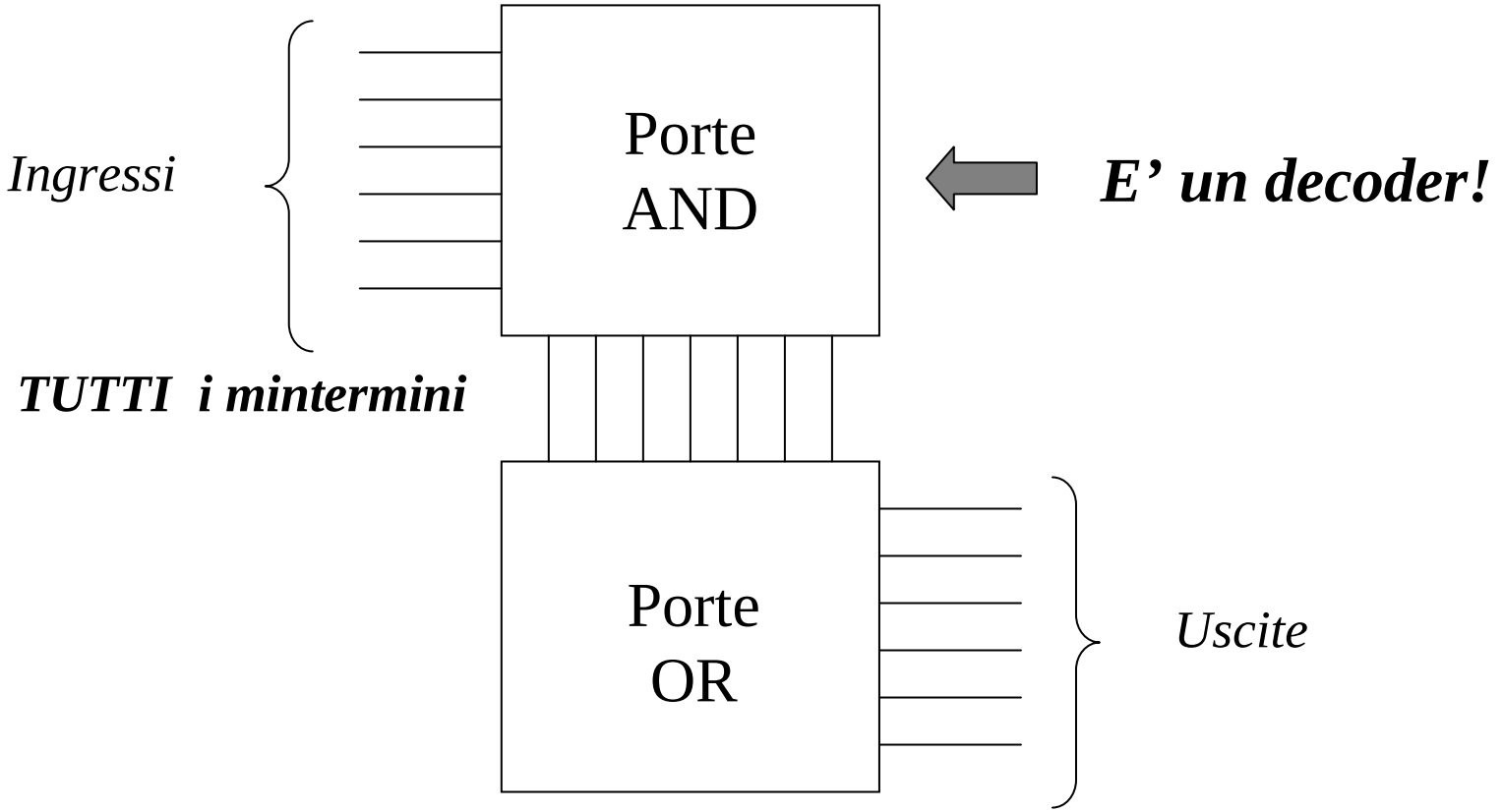
- Una ROM è un altro tipo di logica strutturata
- Ha un insieme di locazioni il cui contenuto è fisso
- I valori degli **ingressi** possono essere visti come **indirizzi,** mentre i valori delle **uscite,** in corrispondenza di certi valori degli ingressi, corrispondono al **contenuto di una locazione**
- Una ROM codifica *m* funzioni ad *n* ingressi: *n* linee di indirizzo con elementi ampi *m* bit ciascuno
- Rispetto alla PLA, è un dispositivo **completamente decodificato**



Dimensione = numero delle celle * ampiezza (in bit) della cella =

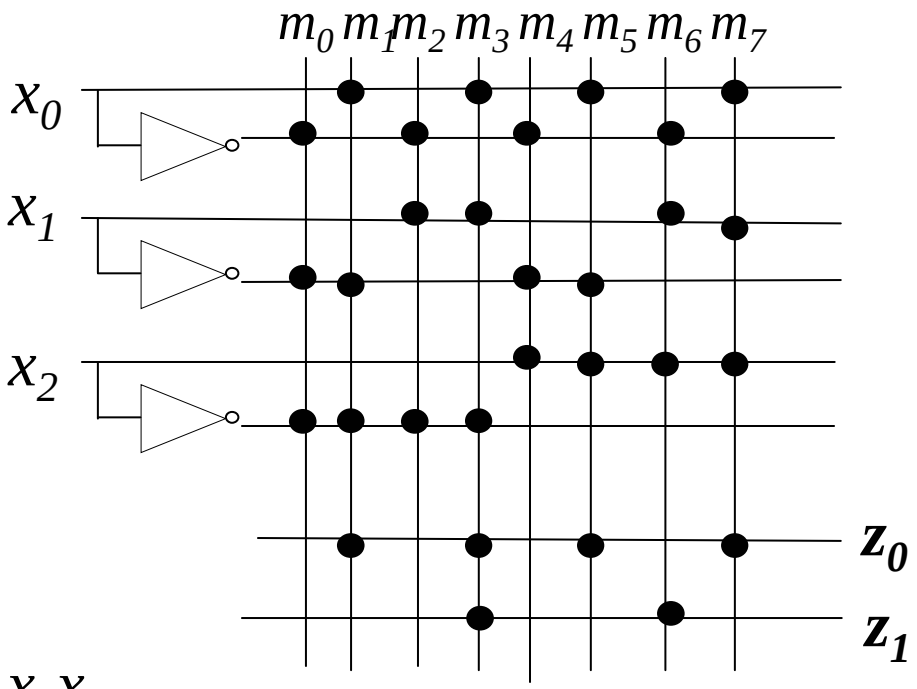
= $\mathbf{2^n * m}$

ROM e PLA


Ingressi
Porte
AND
E' un decoder!
TUTTI i mintermini
Porte
OR
Uscite

Torniamo all'esempio precedente

x_2	x_1	x_0	z_1	z_0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	0	1



$$z_0 = \overline{x}_2\overline{x}_1x_0 + \overline{x}_2x_1x_0 + x_2\overline{x}_1x_0 + x_2x_1x_0$$

$$z_1 = \overline{x}_2x_1x_0 + x_2x_1\overline{x}_0$$

PLA vs. ROM

Vantaggio PLA: dimensioni contenute

- è sufficiente tenere conto dei soli elementi della tabella di verità che producono un valore vero per almeno un'uscita;
- i termini prodotto della PLA possono essere utilizzati in più uscite
- invece, la ROM è completamente codificata
  (m bit per ognuna delle 2^n righe)

⇒ Numero di elementi ROM esponenziale rispetto a n
Numero di elementi PLA cresce molto più lentamente

Vantaggio ROM: maggior facilità di cambiamento

- dati n e m, le dimensioni della ROM non cambiano al variare della funzione logica: se funzione logica cambia, basta modificare il contenuto della ROM

RETI LOGICHE COMBINATORIE DI USO COMUNE

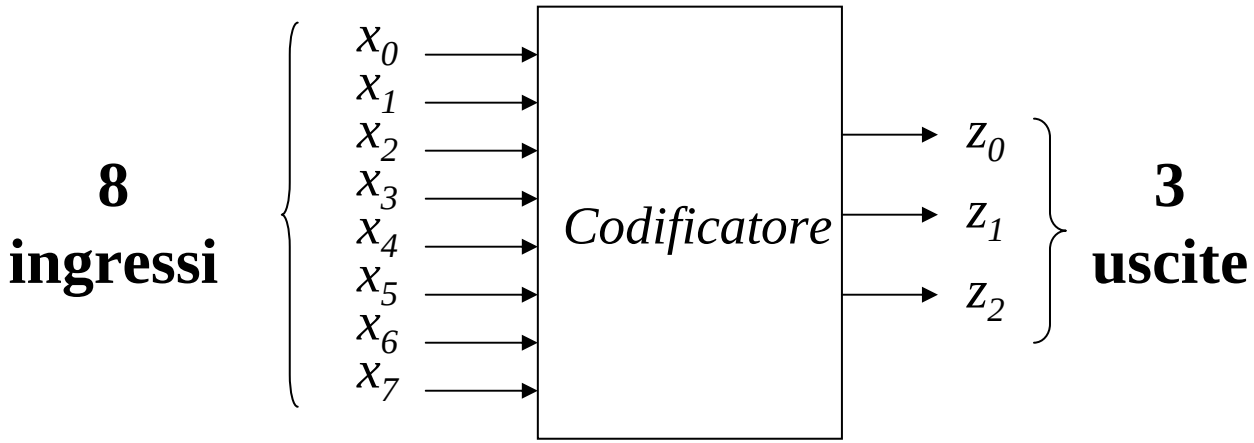
- Codificatori
- Decodificatori
- Multiplexer
- Demultiplexer

Codificatori

- Codificatore *n-a-m*: rete combinatoria con *n* linee di ingresso ed *m* linee di uscita (con $m = \lceil \log_2(n) \rceil$)
- **Combinazioni ammesse**: quelle che contengono ***un solo uno*** ed ***n-1 zeri*** (*n* combinazioni in tutto)
- L'**uscita** del codificatore è la **codifica binaria dell'indice *i*** dell'unica linea di ingresso attiva

$$i = z_{m-1}2^{m-1} + z_{m-2}2^{m-2} + \ldots + z_1 2^1 + z_0 2^0$$

Esempio: codificatori 8-a-3



- Supponiamo che in ingresso sia attiva la linea 4
- Poiché $4_{10} = 100$
- Le 3 uscite saranno $z_0 = 0$, $z_1 = 0$ e $z_2 = 1$

Codificatori: tabella di transizione

- **Supponiamo *n*=4, *m*=2**

x_3	x_2	x_1	x_0	z_1	z_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$z_0 = \overline{x_3}\,\overline{x_2}\,x_1\,\overline{x_0} + x_3\,\overline{x_2}\,\overline{x_1}\,\overline{x_0}$$

$$z_1 = \overline{x_3}\,x_2\,\overline{x_1}\,\overline{x_0} + x_3\,\overline{x_2}\,\overline{x_1}\,\overline{x_0}$$

- in questo caso ogni configurazione “non ammessa” produce in entrambe le uscite il valore nullo

Codificatori: tabella di transizione

- **Supponiamo *n*=4, *m*=2**

x_3	x_2	x_1	x_0	z_1	z_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

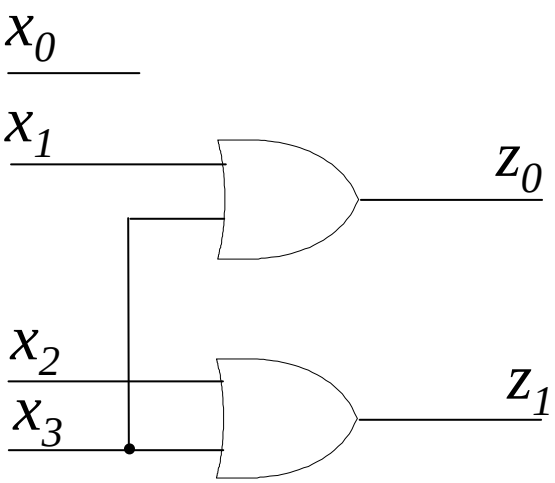
$$z_0 = \overline{x_3}\,\overline{x_2}\,x_1\,\overline{x_0} + x_3\,\overline{x_2}\,\overline{x_1}\,\overline{x_0}$$

$$z_1 = \overline{x_3}\,x_2\,\overline{x_1}\,\overline{x_0} + x_3\,\overline{x_2}\,\overline{x_1}\,\overline{x_0}$$

- in questo caso ogni configurazione “non ammessa” produce in entrambe le uscite il valore nullo
- potremmo però notare che se x_i è attivo, tutti gli altri sono nulli

$$\Rightarrow \quad z_0 = x_1 + x_3 \qquad z_1 = x_2 + x_3$$

Realizzazione circuitale



- In questo caso, le configurazioni non ammesse portano ad uscite indefinite (p.es. x_1=1 porta a z_0=1 in ogni caso)
- Per introdurre un criterio, si può considerare un ordine di priorità tra gli ingressi. Per esempio $x_3 > x_2 > x_1 > x_0$:
  - se x_3=1 l'uscita è 11 a prescindere dagli altri ingressi
  - se x_2=1 (e x_3=0) l'uscita è 10 in ogni caso
  - ecc. ecc.

Codificatori: tabella di transizione

- **Supponiamo *n*=4, *m*=2**

x_3	x_2	x_1	x_0	z_1	z_0
0	0	0	0	0	0
x	x	x	1	0	0
x	x	①	⓪	0	1
x	1	0	0	1	0
①	⓪	0	⓪	1	1

$$z_0 = x_1 \overline{x_0} + x_3 \overline{x_2}\,\overline{x_0}$$

NB: $x_0 > x_1 > x_2 > x_3$

Codificatori: tabella di transizione

- **Supponiamo n=4, m=2**

x_3	x_2	x_1	x_0	z_1	z_0
0	0	0	0	0	0
x	x	x	1	0	0
x	x	1	0	0	1
x	1	0	0	1	0
1	0	0	0	1	1

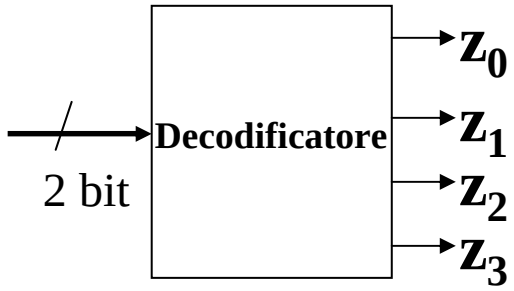
$$z_0 = x_1 \overline{x_0} + x_3 \overline{x_2}\,\overline{x_0}$$

$$z_1 = \overline{x_1}\,\overline{x_0}\,(x_2 + x_3) = \overline{x_1}\,\overline{x_0}\,x_2 + \overline{x_1}\,\overline{x_0}\,x_3$$

NB: $x_0 > x_1 > x_2 > x_3$

Decodificatori

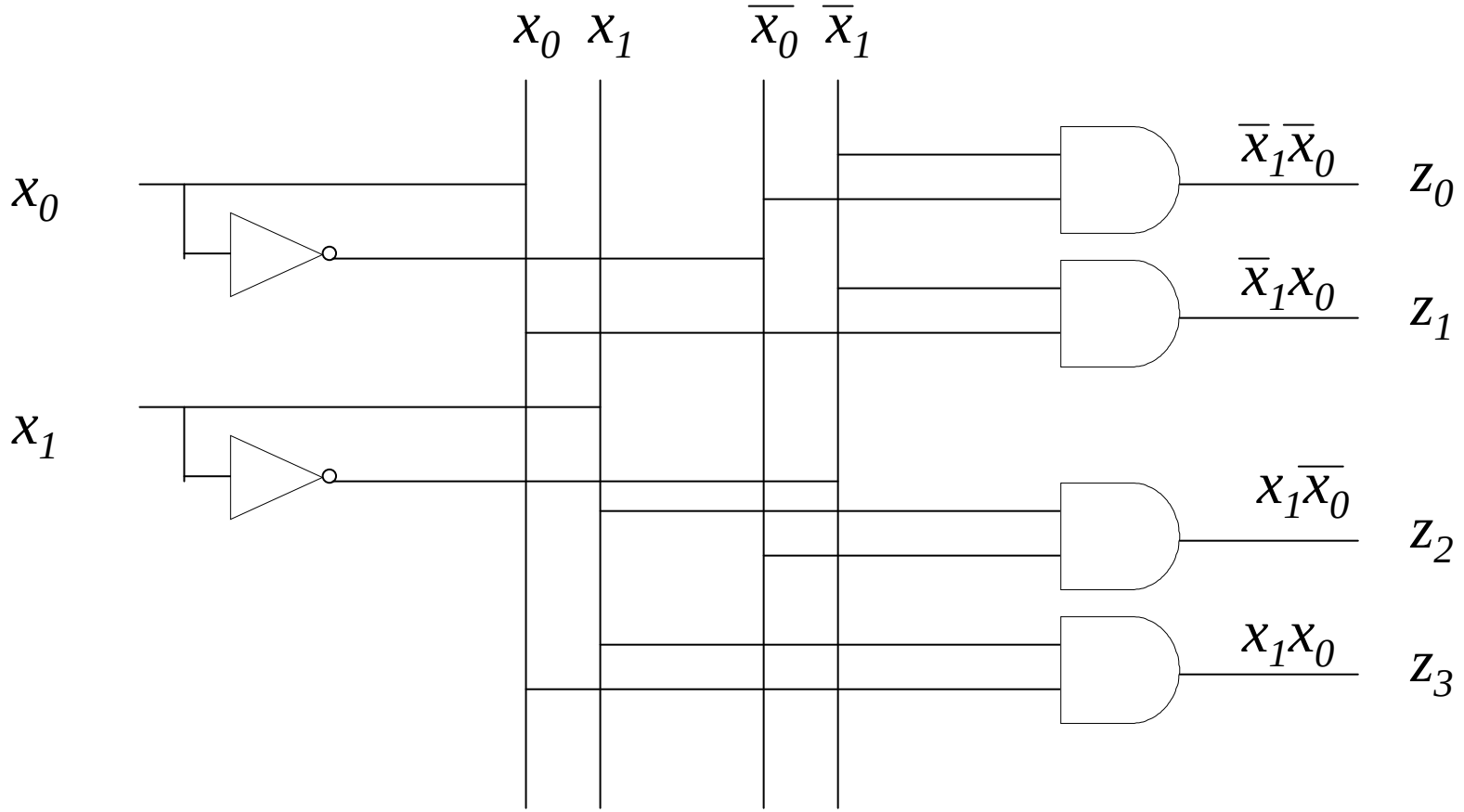
- Ingresso di *n* bit e 2^n uscite
- Per ciascuna combinazione degli ingressi, una sola uscita assume il valore 1 mentre le altre assumono valore 0



Se il valore dell'ingresso è *i* allora l'uscita z_i sarà vera e tutte le altre false

x_1	x_0	z_3	z_2	z_1	z_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Esempio: Decodificatore 2-a-4



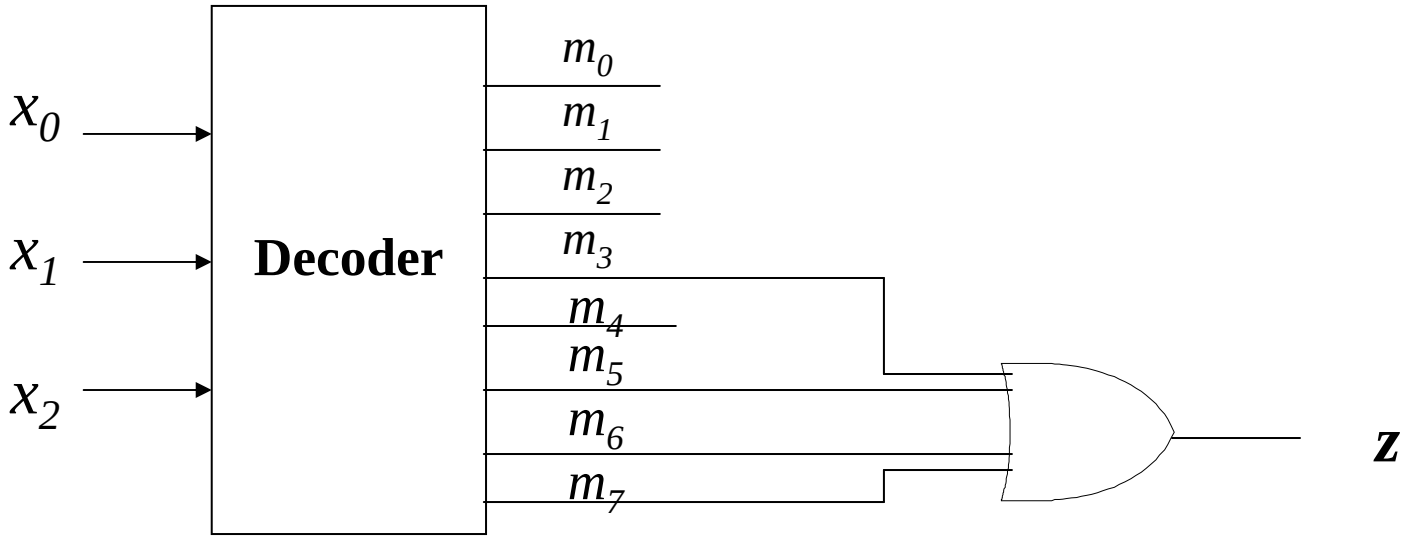
Decodificatori e forma canonica somma di prodotti

- In pratica, un decodificatore fornisce alle sue uscite i mintermini degli ingressi

$$z_0 = m_0 \quad z_1 = m_1 \quad z_2 = m_2 \quad z_3 = m_3 \ldots$$

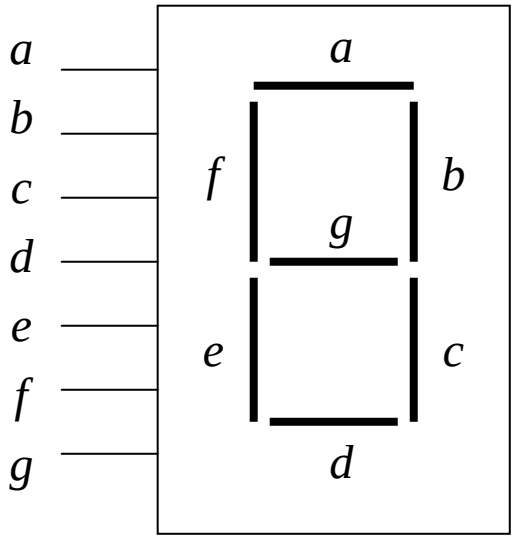
- Quindi, una qualsiasi **forma canonica somma di prodotti** può essere realizzata utilizzando un **decodificatore** e una **porta OR**

Esempio: funzione di maggioranza



$$z = m_3 + m_5 + m_6 + m_7 = \bar{x}_2 x_1 x_0 + x_2 \bar{x}_1 x_0 + x_2 x_1 \bar{x}_0 + x_2 x_1 x_0$$

Esempio: controllo di un display a 7 segmenti



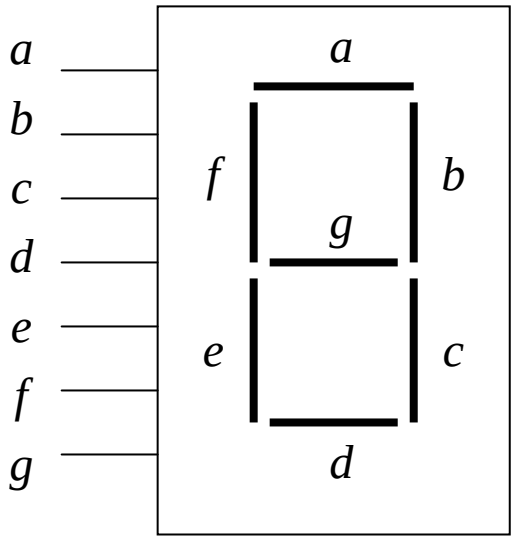
Display a 7 segmenti

- *Ogni segmento è alimentato in modo indipendente dagli altri*
- *Una cifra fra 0 e 9 può essere formata alimentando una parte dei segmenti*
- *La rete combinatoria che comanda il display ha:*

  Tanti ingressi quanti sono i bit del codice (4 nel ns. caso)

  7 uscite (che corrispondono ai segmenti)

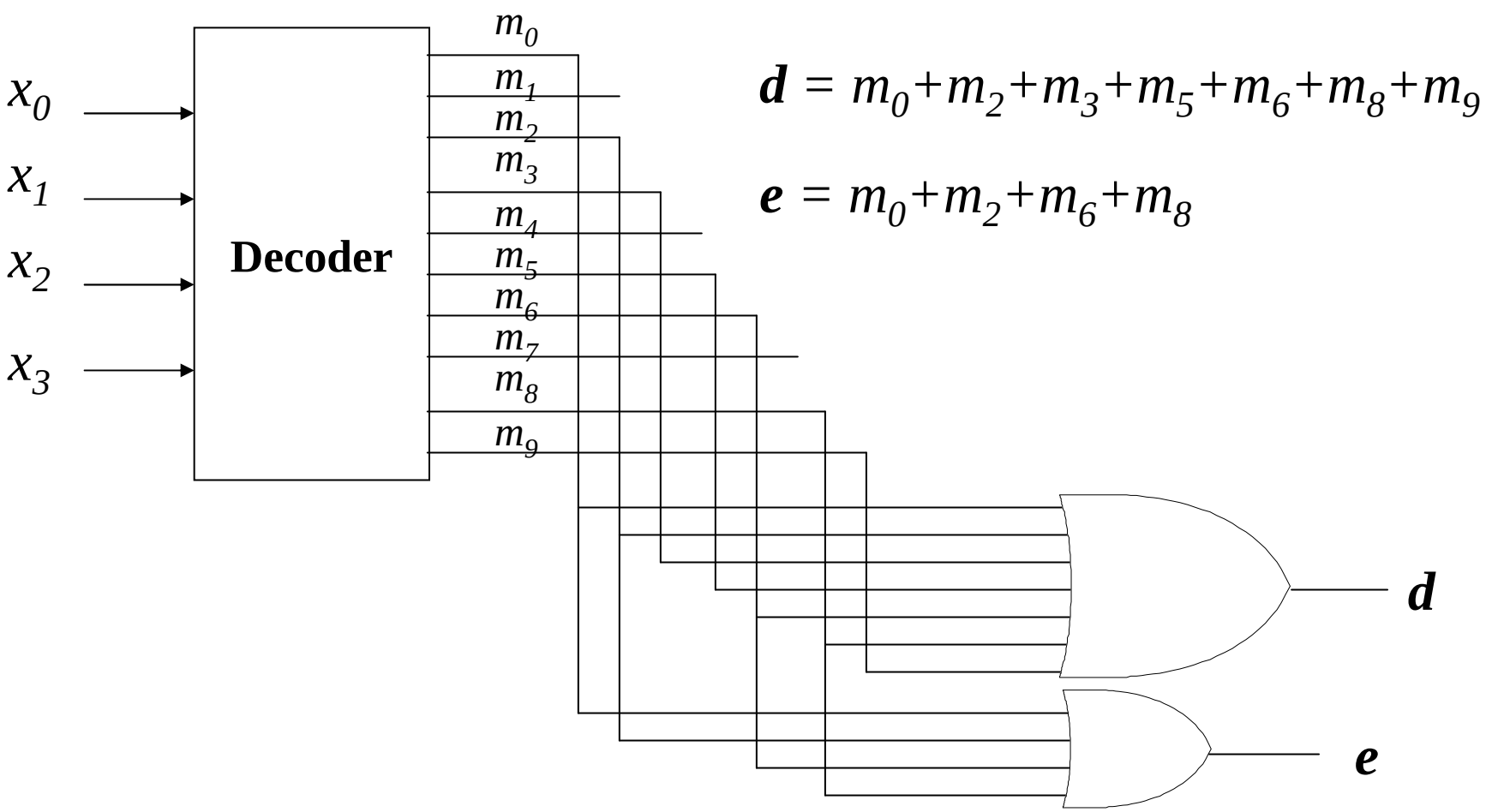
Tavola di verità



Display a 7 segmenti

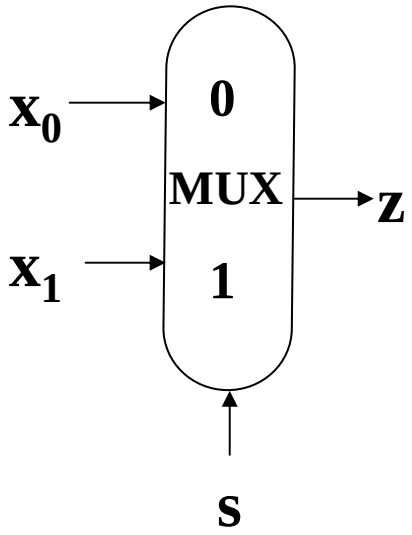
$x_3x_2x_1x_0$	***a***	***b***	***c***	***d***	***e***	***f***	***g***
0000	1	1	1	1	1	1	0
0001	0	1	1	0	0	0	0
0010	1	1	0	1	1	0	1
0011	1	1	1	1	0	0	1
0100	0	1	1	0	0	1	1
0101	1	0	1	1	0	1	1
0110	1	0	1	1	1	1	1
0111	1	1	1	0	0	0	0
1000	1	1	1	1	1	1	1
1001	1	1	1	1	0	1	1

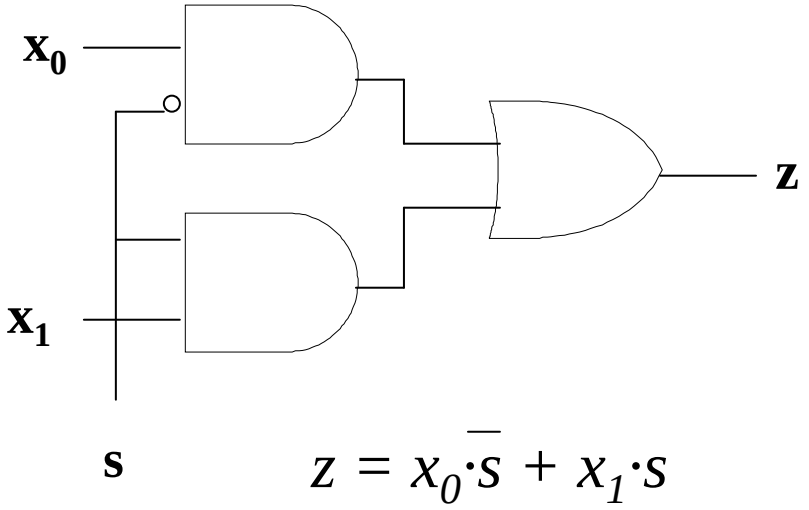
Realizzazione per i segmenti *d* ed *e*



Multiplexer

- Detto anche ***selettore***: la sua uscita è uguale a uno degli ingressi, scelto mediante un segnale di controllo (S)

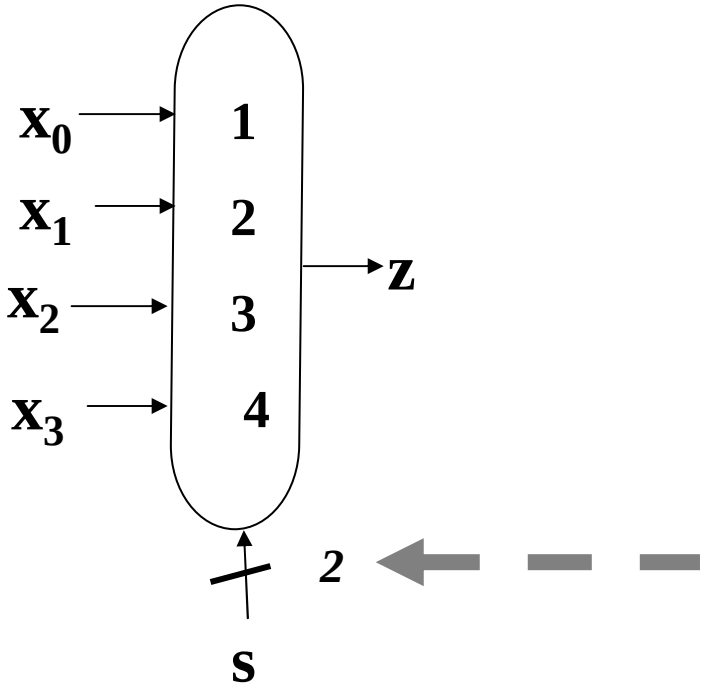




$$z = x_0 \cdot \overline{s} + x_1 \cdot s$$

Es. Multiplexer a 2 vie

Esempio: multiplexer a 4 vie



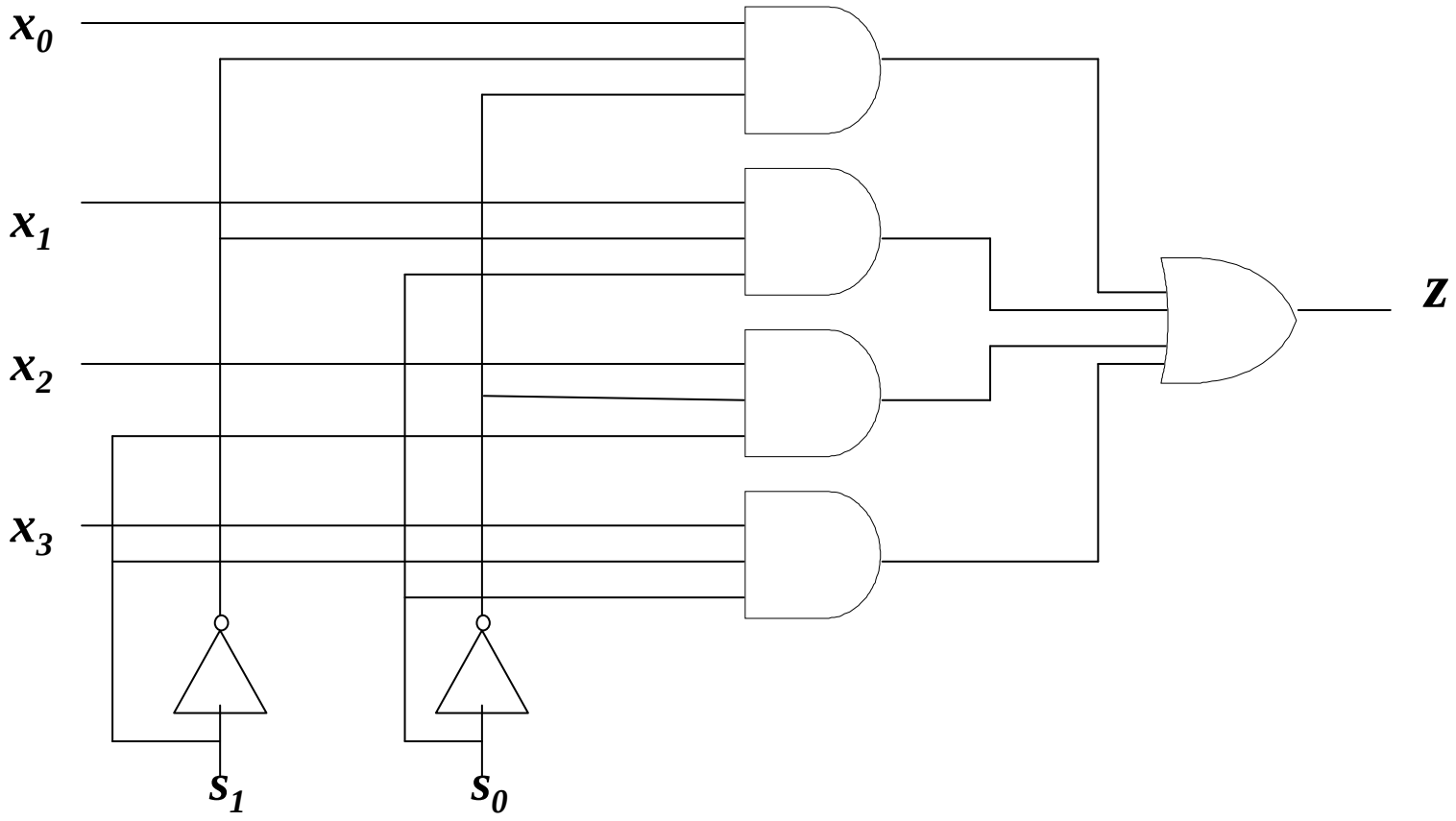
Esempio: multiplexer a 4 vie

- 4 **ingressi di dato**: x_0, x_1, x_2, x_3
- 2 **ingressi di selezione**, s_0 e s_1, t.c. 00 → x_0 , 01 → x_1, 10 → x_2, 11 → x_3

s_1	s_0	x_3	x_2	x_1	x_0	z
0	0	x	x	x	0	0
0	0	x	x	x	1	1
0	1	x	x	0	x	0
0	1	x	x	1	x	1
1	0	x	0	x	x	0
1	0	x	1	x	x	1
1	1	0	x	x	x	0
1	1	1	x	x	x	1

Ingressi di selezione *Ingressi di dato* *uscita*

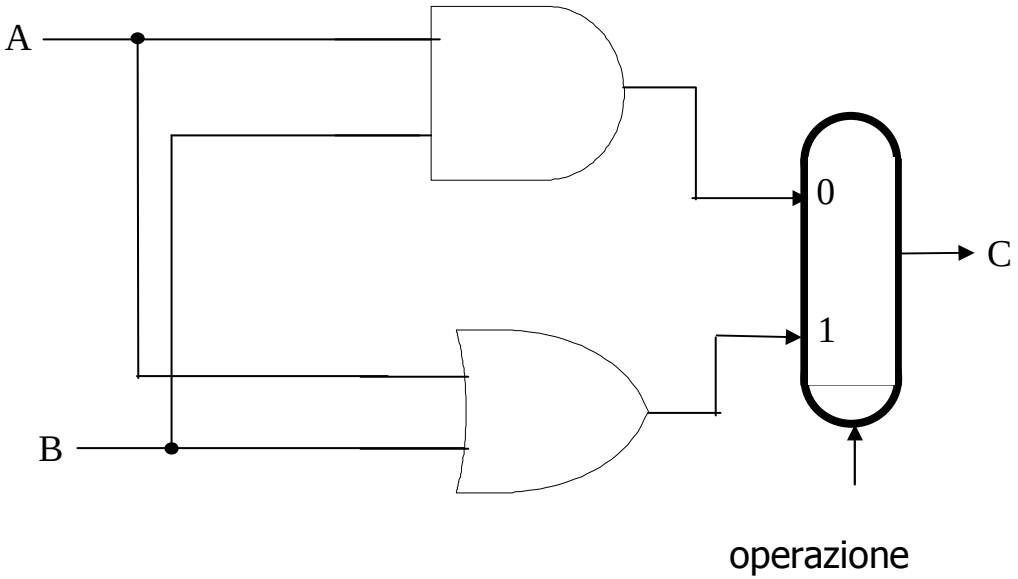
Realizzazione circuitale



Applicazioni del Multiplexer

- Il multiplexer viene in genere usato per la **selezione di una linea di input** (con n linee di dato sono necessarie $\lceil \log(n) \rceil$ linee di selezione)
- Un'altra applicazione è come **convertitore di dati da parallelo a seriale**: avendo in input 8 bit di dati e avendo linee di controllo a 3 bit, queste ultime sono in grado di serializzare in output gli 8 bit instradandoli uno alla volta (000, 001, …, 111)
- Un altro uso nelle reti combinatorie è la selezione di una uscita tra quelle corrispondenti a più funzioni (vedi lucido successivo)

un esempio: calcola o l'AND o l' OR

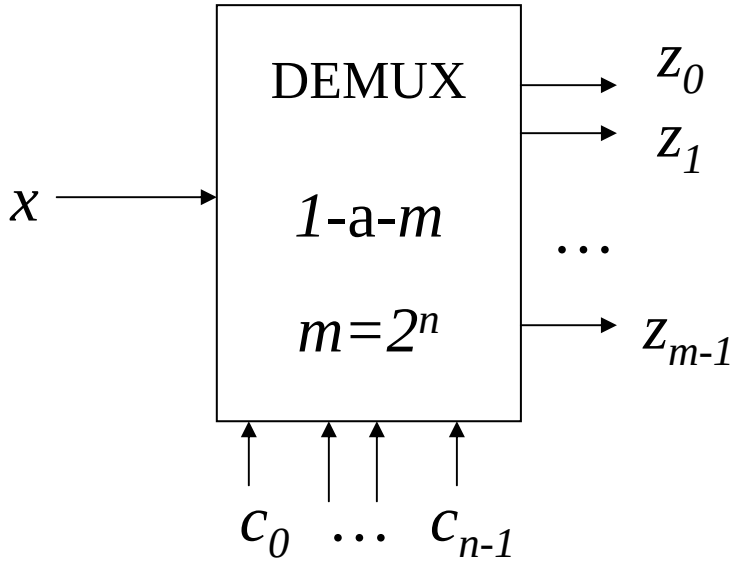


If operazione==0,
C=A+B
else C=A*B

operazione	***A***	***B***	***C***
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Demultiplexer

- Si potrebbe chiamare ***distributore***
- Il suo compito è quello di instradare un singolo input in una delle 2^n linee di output a seconda del valore delle n linee di controllo.



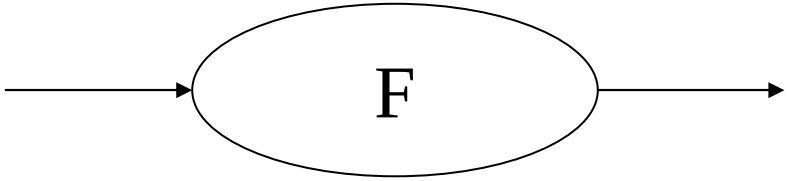
Esempio: demultiplexer 1-a-4

c_1	c_0	x	z_0	z_1	z_2	z_3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

Ingressi di controllo ***Ingressi di dato*** ***uscita***

$z_0 = \overline{c}_1 \overline{c}_0 x$ $z_1 = \overline{c}_1 c_0 x$ $z_2 = c_1 \overline{c}_0 x$ $z_3 = c_1 c_0 x$

TEMPO DI PROPAGAZIONE DI RETI LOGICHE COMBINATORIE



Dal momento in cui l'ingresso è valido al momento in cui l'uscita è valida trascorre un certo intervallo temporale: